TECHNION, ISRAEL INSTITUTE OF TECHNOLOGY
Department of Mechanical Engineering

Robot Navigation

Part #2 of Project

Spring 2025 Issued: July 09, 2025

Due: September 07, Lady Davis 640, 15:00-16:00.
Submission: student pairs (less desirable: student triplets).
Bring: Printed project report. No laptop demos.

Path Planning for a Polygonal Robot in a Polygonal Environment

You already discretized the (x, y, θ) configuration space of the apartment depicted in Figure 2 in Part #1 of the project. The outer wall is the union of $\mathcal{B}_{0_1}, \mathcal{B}_{0_2}, \mathcal{B}_{0_3}, \mathcal{B}_{0_4}$. The interior walls are $\mathcal{B}_1, ..., \mathcal{B}_5$. The doors are $\mathcal{B}_6, \mathcal{B}_7, \mathcal{B}_8$. The coordinates of $\mathcal{B}_2$: $(17, 17), (18, 17), (18, 29), (17, 29)$. The coordinates of $\mathcal{B}_{0_1}$: $(0, 29), (32, 29), (32, 30), (0, 30)$.

For the scene in Figure 2: 1) Pick one path-planning method from the ones offered below. 2) Implement it using the c-space code generated in Part #1 of the project. 3) Submit *a printout showing the silhouette of the robot as it travels from the start to the target specified in Figure 2.* The style should be similar to the one shown in Figure 3

Grid resolution: The ad-hoc resolution of $32 \times 32 \times 32$ is probably too coarse for the θ axis, for the following reason. The discretized (x, y, θ) space is initially zeroed (all cells are empty), then only the *boundaries* of the c-obstacles are marked. In a coarse grid, it may happen that a c-obstacle edge moves between two consecutive θ layers such that a true free cell has an immediate neighbor in the other layer that lies strictly in the interior of a c-obstacle. Such a cell is unmarked as an obstacle, see Figure 1. You must first derive a criterion for choosing the θ resolution so that no c-obstacle edge moves more than one cell between two consecutive θ layers.

The following are your options for a path planner:

1. A^***:** Consider the (x, y, θ) grid to be a graph whose nodes are the centers of the *free* grid cells, and whose edges connect immediate neighbors in the grid. Do A^* search for the target in this graph. Draw the collection of nodes "closed" by the A^*, verify that an ellipsoid is generated.

   A^* **question:** A disc robot navigates in an *unknown* planar environment using a position sensor and a local obstacle detection sensor. Explain how A^* can be used to navigate the robot *on-line,* by scanning neighboring cells in the discretized (x, y) grid of the environment. (The on-line path requires physical motion of the robot between the current x_{best} to the next x_{best}. Plot a graph or table of actual path length traveled compared with optimal off-line path length of robot center point.)

2. **Depth First Search:** Depth First Search is another classical graph-search algorithm, called DFS. Read the algorithm description in the appendix of Latombe's book, and implement it on the grid-cells graph described above.

3. **3-D visibility graph:** You will need only the *coordinates* of the c-obstacle vertices. 1) Construct the reduced Visibility Graph for each θ slice. 2) Add edges between vertices on consecutive θ layers that correspond to the same contact in workspace. 3) Do graph search such as A^* on the resulting graph.

   Extra question: A rectangular robot navigates in an *unknown* planar environment using position sensor and local obstacle detection sensors. Explain how an on-line constructed 3-D visibility graph (based on currently known environment) can be used to navigate the robot *on-line,* based on the position and obstacle detection sensors.

4. **Potential functions:** 1) Pre-compute the shortest path navigation function in the discretized (d_x, d_y, θ) configuration space of the environment, using a simple Breadth First Search (see algorithm description in the appendix of Latombe's book). 2) Apply the best-first search method on this potential function.

   Extra question: A disc robot navigates in an *unknown* planar environment using position sensor and local obstacle detection sensors. Explain how an on-line constructed shortest path navigation function (based on currently known environment) can be used to navigate the robot *on-line,* based on the position and obstacle detection sensors.

5. **Rapidly exploring search trees:** This is the *newest* robot navigation method, called *RRT*. It was not covered in the course due to lack of time. The *RRT* method builds a collection of explored c-space cells as an expanding graph, one new cell in each step. The search graph G is initialized to the start cell S. In each step, the robot *randomly* selects a new free cell in c-space, and connects to the closest cell in the current graph G. The process ends when the target T is selected and connected to the graph G. Finally, a shortest path computation on G gives the path from S to T.

   Newest *RRT* methods: Please check *Bi–RRT* and *RRT–Connect*.

   Extra question: A disc robot navigates in an *unknown* planar environment using position sensor and local obstacle detection sensors. How can *RRT* be adapted to the on-line setting? Practical issues: execution time comparable to A^* ?

6. Other ideas are most welcome, but *email Prof. Rimon first.*

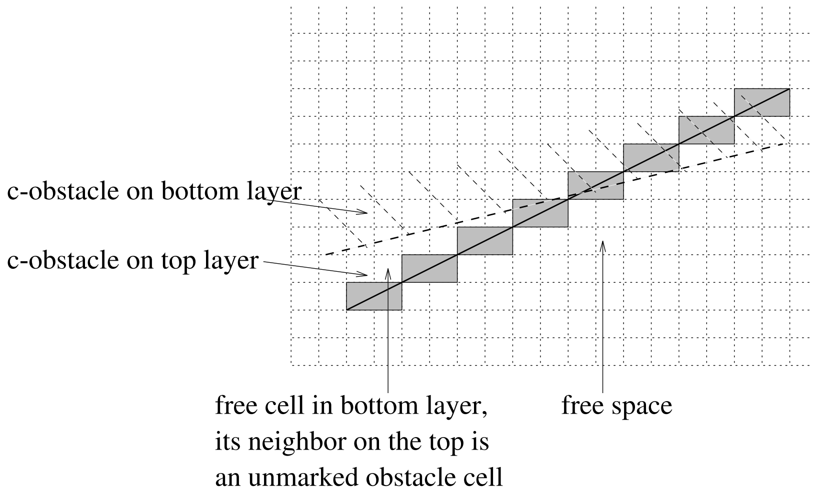

Figure 1: θ resolution consideration.

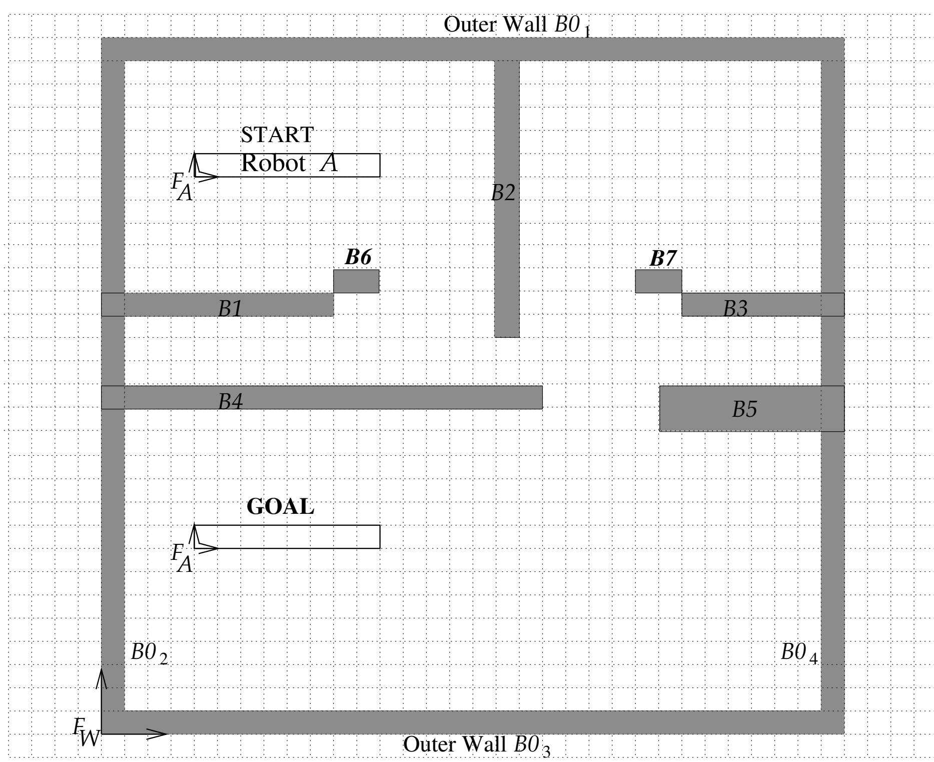

Figure 2: The Bed Movers Problem.

ניווט גוף קשיח

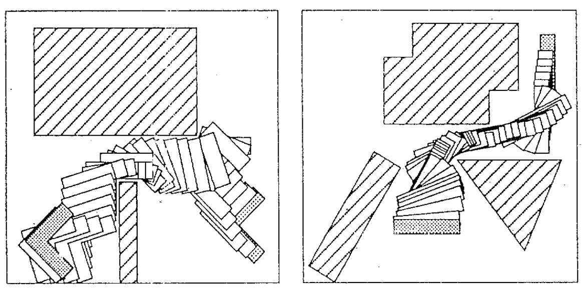

Figure 3: Example of printout style

Figure 3: Example of printout style

3